TALLINN UNIVERSITY OF TECHNOLOGY

School of Information Technologies

Kirill Grišin 193712IAIB

Computer Vision Algorithms for Fast Object Detection on Gaming Equipment

Bachelor's thesis

Supervisor: Järv, Priit
Ph.D

Vassiljev, Roman
MSc

Tallinn 2022

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Kirill Grišin 193712IAIB

Tehisnägemise algoritmid objektide kiireks tuvastamiseks mänguseadmetes

Bakalaureusetöö

Juhendaja: Järv, Priit
Ph.D

Vassiljev, Roman
MSc

Tallinn 2022

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Kirill Grišin

30.05.2022

Abstract

The aim of this work is to investigate new possibly faster ways of detecting the objects on the professional gaming equipment, like roulette wheel, using digital image analysis, and create a prototype of the possible algorithm.

The contribution of the work is improvement of the currently used image recognition algorithm in terms of accuracy and speed, as this software is used in production and the flaw rate is crucial. The currently existing algorithm is the part of internal (Playtech plc) software.

In the course of the work, the theoretical material was gathered and analyzed, the basics of problematic topic were introduced, and the possible algorithm prototype was created. Using the internal software, the prototype was implemented and tested.

As the work was finished and algorithm was tested, it is possible to state that goals were achieved. Ways of the performance improvement were found, and total algorithm efficiency has raised.

This thesis is written in English and is 31 pages long, including 7 chapters, 17 figures and 1 table.

Annotatsioon

Antud töö eesmärgiks on digitaalse pildianalüüsi abil uurida uusi võimalikke kiiremaid viise objektide tuvastamiseks professionaalsetel mänguseadmetel, nagu ruletiratas, ning luua võimaliku algoritmi prototüüp.

Töö panus on praegu kasutusel oleva pildituvastusalgoritmi täiustamine täpsuse ja kiiruse osas, kuna seda tarkvara kasutatakse tootmises ja vigade määr on oluline. Praegu olemasolev algoritm on osa Playtech sisetarkvarast.

Töö käigus koguti ja analüüsiti teoreetiline materjal, tutvustati probleemse teema põhitõdesid ning loodi võimalik algoritmi prototüüp. Sisetarkvara abil teostati prototüüp ja seda testiti.

Kuna töö oli valmis ja algoritm testitud, siis võib väita, et eesmärgid saavutati. Leiti jõudluse parandamise viisid ja kogu algoritmi efektiivsus on tõusnud.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 31 leheküljel, 7 peatükki, 17 joonist, 1 tabelit.

List of abbreviations and terms

DPI	Dots per inch
IA	Department of Computer Systems
2D	Two-dimensional
3D	Three-dimensional
CPU	Central processing unit
GPU	Graphics processing unit
fps	Frames per second
kb/s	Kilobytes per second
ms	Milliseconds
cm	Centimeters
m	Meters
GHz	Gigahertz
GB	Gigabyte
RAM	Random access memory
DDR	Double Data Rate
OS	Operational system
px	Pixels

Table of contents

List of figures

List of tables

1 Introduction

Professional equipment for entertainment purposes, more specifically in the field of virtual casino games with real dealers, is often not fully automated and some part of its possible functionality must be performed by human. For example, result determination is a part of such functionality. In roulette games, the dealer usually must manually determine and confirm round result. However, the amount of rounds that dealer daily host can reach up to multiple hundreds.

Such intense load is not something rare nowadays in the production, therefore dealers are being equipped with additional tools, which cover the part of their routine responsibilities. Some part of such responsibilities can be described as result determination in real-time, which can be fully automated on the software side. This software is commonly described as image recognition software.

Image recognition software uses computer vision algorithms for different cases, for example, for separating the objects on the image (like roulette ball) or color difference comparison (for determining a sector in Spin-A-Win game). Although, these algorithms often are not fully optimized, as they contain in their implementation a large number of ineffective parts.

1.1 Existing solutions

1.1.1 SET-Production Cyclops

One of alternative solutions of the specified problem is roulette displays manufactured by SET-Production company, Finland.

Cyclops is a family of video capturing devices to read winning number from roulette wheel. It is not only state-of-art ball detection technology which can read winning number within a second, but also powerful secure tool which makes you sure that every spin on the roulette is correct. All types of roulette wheel with colored or monochrome (black, green, red, orange) pockets are supported. The state of wheel and ball, all game events permanently transmitted into local network. [6]

This solution was tested before in Playtech company and received mostly negative reviews.

The main negative part of this solution is performance, as, during tests, a lot of errors were caught, as well as overall responsiveness was critically low. This solution does not provide the source code, therefore it is impossible to debug or bring changes to solution’s behavior. Another negative side of this solution is cumulative cost of the setup per one roulette table. Each display requires additional equipment that company offers separately. Such pricing policy raises the cost of each roulette table in multiple times.

1.1.2 Gambee Supreme Roulette

Another alternative solution that was investigated are Supreme Roulette series manufactured by Gambee company, Slovenia.

The whole solution represents a large setup with multiple (up to 10) seats and roulette wheel. Each seat includes 27 inches display with pre-installed software. The most part of gaming process is digitalized: players can place their bets using provided displays with an interface, as well as track the ball movement on the roulette wheel, which has central position among all seats. [9]

Such solution has very specific use-cases, as it is complete roulette setup with pre-defined options, seats and algorithms. This solution has also closed source code, therefore the solution partly has the same problems that are described in SET-Production Cyclops analysis (see section 1.1.1). Single instance of this solution also has a high price, that is even higher than SET-Production Cyclops setup price.

1.2 Goals

After analyzing alternative solutions, author decided to investigate the ways of improvement of an existing algorithm. For this work such goals were defined, as:

- Gather and analyze theoretical material on the „Image recognition“ and related topics.
- Describe a new algorithm prototype, which parts can be integrated into internal recognition software.
- Implement the prototype using the internal image recognition framework.
- Run multiple performance tests using real production situations and analyze the prototype effectiveness and accuracy realtive to the current internal algorithm.

2 Digital image concept

2.1 What is an image?

A computer system processes an image in a very specific way. The system represents an image as a set of spatial and intensity information. The 2D image *I(m, n*) can be represented as a response of some sensor as an 2D array of fixed positions (m=1,2,...,M; n=1,2,...,N) in 2D Cartesian coordinates and is derived from continuous spatial signal I(x, y) through a sampling process named discretization.

The indices m, n represent respectively rows and columns of a digital image. Each pixel of an image can be referred to by its 2D index (m, n).

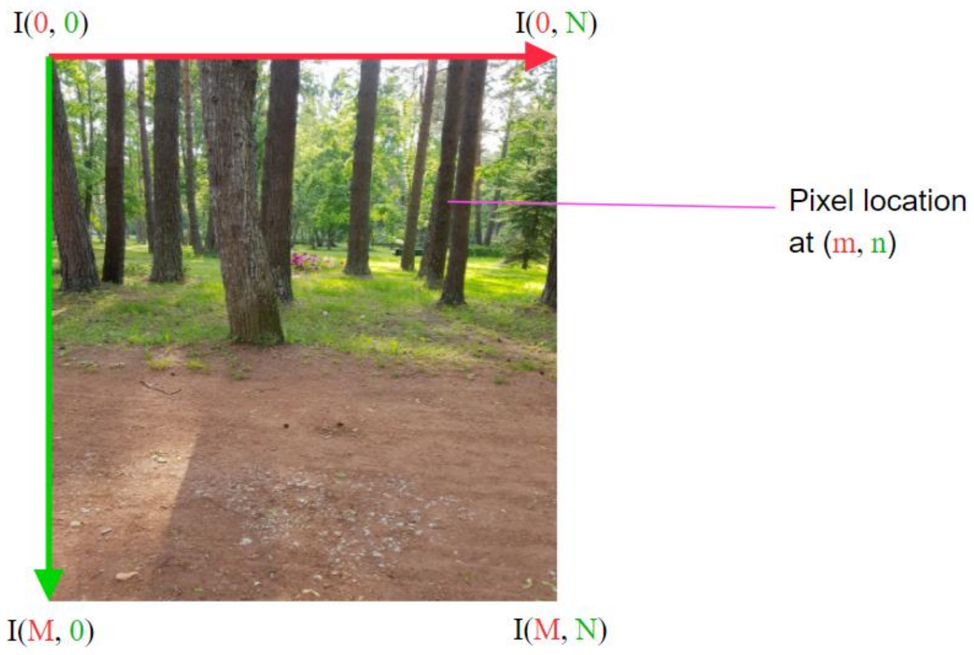


Figure 1. 2D Cartesian coordinate space of an (M, N) digital image.

2.2 Image color representation

The pixel value in most digital images correspond to some physical response in real 2D space. An image contains one or more color channels that define the intensity or color at a specific pixel location *I(m, n)*. In the simplest case, each pixel location at *(m, n)* contains a single numerical value, which represents the signal level from sensor. The system uses a special set of values named color map to convert the values contained by pixel locations to an actual image on the display.

2.2.1 Greyscale

The most common color map is the greyscale map. This map takes on input a sensor value and gives in the output one of the shades of grey color (from black color to white color) depending on the signal intensity of the given value.

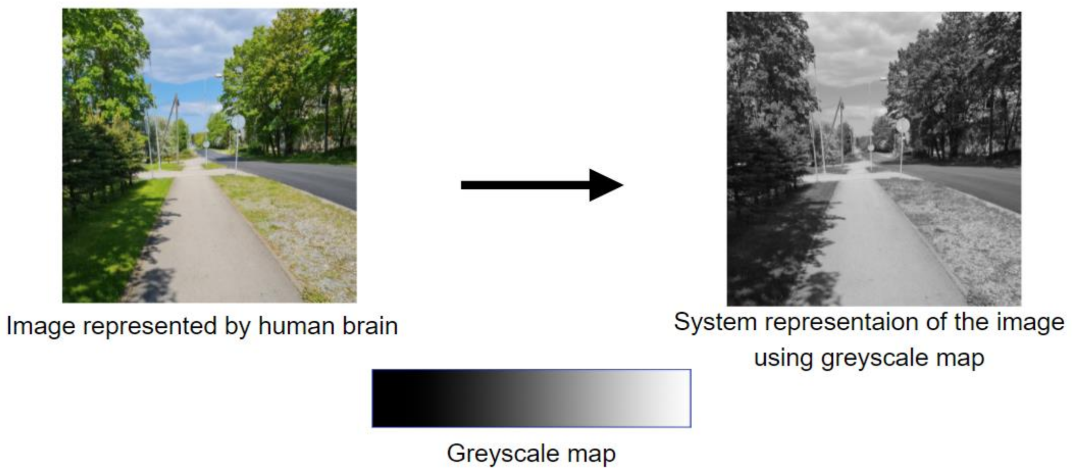


Image represented by human brain
System representaion of the image using greyscale map
Greyscale map

Figure 2. Image representation using greyscale color map.

2.2.2 True color images and RGB

The more complex representations of an images are true color image, which use the full spectrum of colors. The true color image consists of three 2D planes, where a plane can be considered as a set of values that correspond to the one of the three coordinates of the RGB (red-green-blue) vector. Each plane contains values that are used to represent only one coordinate. Red, green, and blue colors are basis colors, meaning that each pixel color of the digital image is calculated using only red, green and blue color in different proportions.

2.2.3 HSV

Another widely used representation of color is HSV (hue, saturation, value). This model describes a color in terms of its saturation and brightness.

Any digital image can be described by a system using both RGB and HSV colour models.

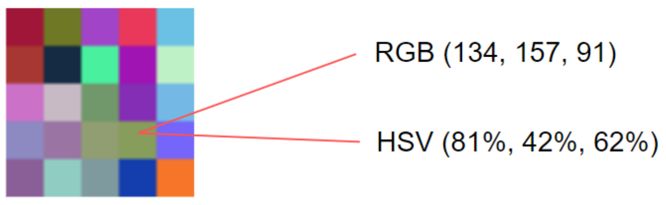


Figure 3. Pixel colour representation using different colour models.

2.3 Pixel conception

The pixel represents the smallest element of a digital image. It is indexed as *(x, y)*, where *x* and *y* are pixel coordinates relative to the image origin. Pixels contain numerical values that are basic units of information within the image itself. In common case, a pixel can be described as a small point within the grid of pixels, the color of which depends on the light input from the scene.

Pixel numerical values are differently represented in different use cases. For example, a common 2D image, that can be found in the web, consists of pixels, which contain an information in the form of three integers. These integers correspond to three color spaces (RGB) and can have a value in the range from 0 to 255. This range can be described as a corresponding color proportion. For example, a set of values (0, 0, 255) means that the pixel is represented as a solid blue color (zeroes mean that red and green channels are completely empty). Some forms of image information can also have floating-point values (3D, scientific, medical). [1]

Within this thesis, author uses a common pixel model, in which a pixel has three integers as color channels (RGB) information.

3 Image processing

3.1 Pixel operations

The image processing consists of operations on pixels of the digital image. The more complex the processing algorithm is, the more pixel operations it uses. Each operation represents an iteration through all pixels and manipulating with the information that they hold.

Algorithm performance heavily depends on the right choice of operations, since any system, that runs the algorithm, has limited computation resources. For example, a computer system performs such operations using CPU or GPU. GPU performs mathematical operations significantly faster, than CPU, as it has more computation cores deployed on it.

3.1.1 Primitive operations

One of the primitive arithmetic operations (among addition, subtraction and division), the multiplication operation, multiplies each pixel channel value by specified value. If the pixel has, for example, a value of (10, 10, 10) in RGB color space and this value is multiplied by 5, the result of this operation on the given pixel will be (50, 50, 50). In other words, the pixel's intensity grew up in 5 times.

The multiplication operation can be used for masking and culling unnecessary objects from the image (see section 4.2.3.2).

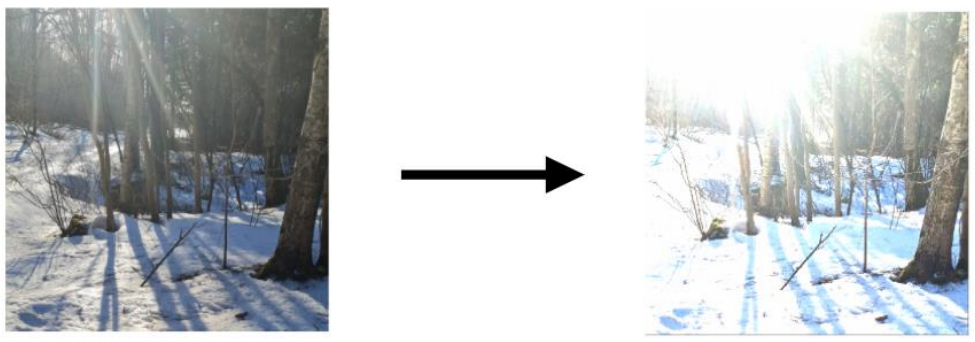


Figure 4. An example of multiplication operation on the complex digital image.

3.1.2 Complex operations

Linear interpolation is a great example of a complex pixel operation. It takes three arguments as its input: a, b and t. The mathematical representation of this operation is:

$$a + (b - a) * t$$

If this operation takes as the first argument a pixel color from one image, and another image pixel color as the second argument – a simple image blending will be achieved.

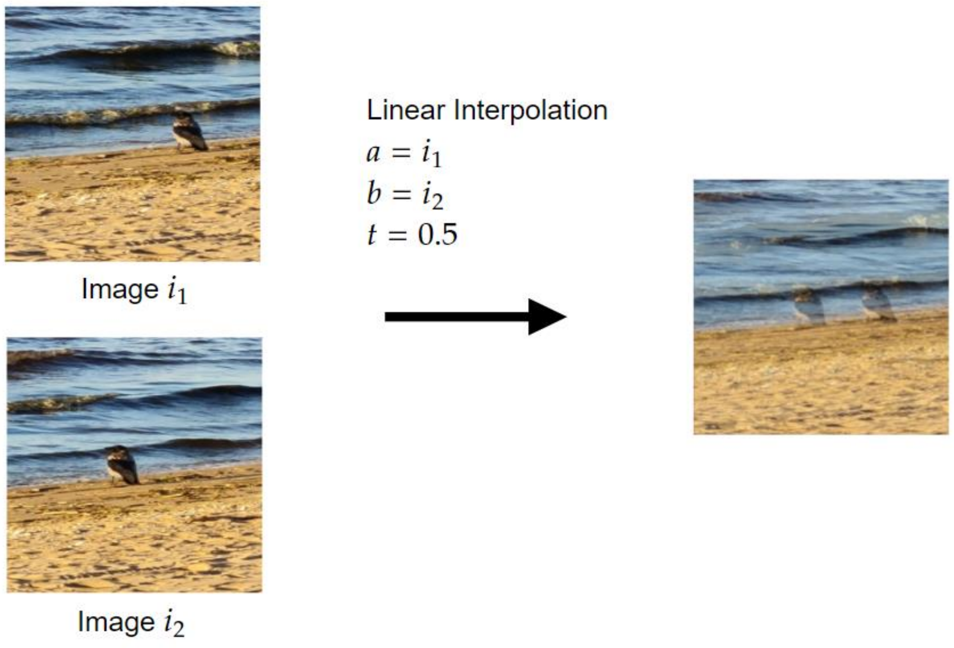


Figure 5. Image blending using linear interpolation.

During this operation, each pixel on the image i_1 has changed its color to a new one that was calculated by taking the original pixel color at position *(m, n)*, a pixel color from image i_2 at the same position *(m, n)*, and interpolating these two colors with the value of $t = 5$. Result of the linear interpolation ($lerp$) of two colors (c_1, c_2), is a new color with channel values: $(lerp(c_1.red,\ c_2.red,\ t),\ lerp(c_1.green,\ c_2.green,\ t),\ lerp(c_1.blue,\ c_2.blue,\ t))$ [5]

Another widely used operation is step function (threshold operation). This function can be used for simple objects segmentation on the digital image. Step function input arguments are: $x, edge$. Mathematically this operation represents a function that is used to compare the x value to the $edge$ value. If $x < edge$, function returns 0, otherwise 1. Applying this function while iterating through all image pixels will cull some of the pixels (set their color value to (0, 0, 0), or solid black color) according to their current value (x value) and given threshold ($edge$ value). [2]

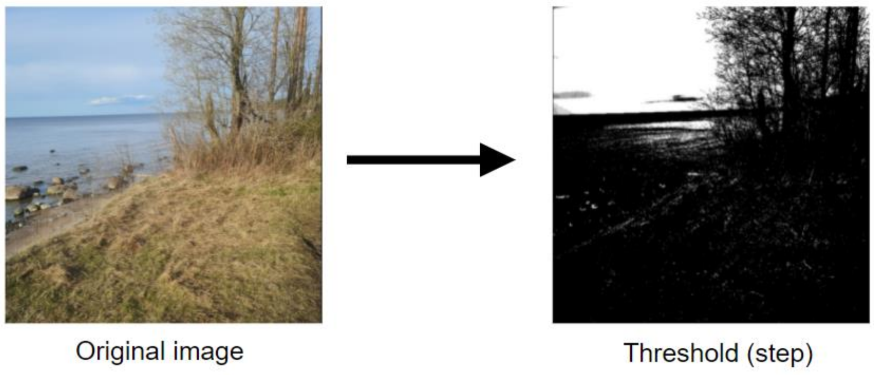


Figure 6. Threshold example using generic image.

Applying threshold to generic complex images usually give an outcome of little significance (as seen on Figure 6), but in case of simple images with static background, it can result in solid results (as shown on Figure 7).

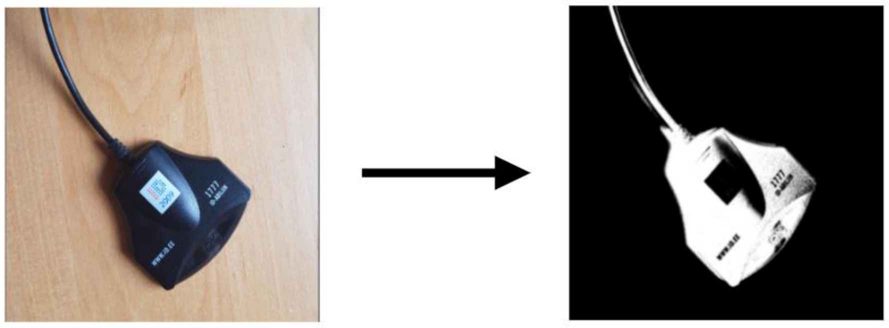


Figure 7. Object detection using simple threshold.

4 Roulette image recognition

4.1 Roulette game rules

Before starting to describe the recognition algorithm, it is necessary to understand the game that this algorithm will be applied to. For this algorithm, the casino game named “Roulette” will be chosen. This game has different rules sets, but thesis author will work with the rules set named “European Roulette”, which can be briefly described as:

- The game consists of a big table, where the roulette wheel and betting area are deployed.
- Players place their bets on specific positions which, as they think, will be winning ones, meaning that the ball will land on the position with the same number. The number is in the range between 0 and 36 (both inclusive).
- Dealer spins the game wheel and launches the ball in clockwise or anticlockwise direction relatively to the wheel.
- After some time, the ball lands on the one of 37 positions, determining the end of the current game round. [7]

4.2 Result determining algorithm

4.2.1 Algorithm purpose

The main purpose of the new result determining algorithm is to determine the position of the ball, so that the pocket, in which the ball has stopped moving, will be known and the game round result will be determined. The game round can last up to 30 seconds, and during this period of time, the algorithm must perform all calculations needed to determine the ball position.

The moment, when the ball stops moving, can be considered as the end point of the algorithm, since the algorithm must determine the ball position and calculate the ball pocket at this very moment.

4.2.2 Algorithm input

The algorithm receives on its input the image from the network stream. This stream is recorded by the camera, placed above the roulette wheel, and is transferred via RTMP (Real Time Messaging Protocol), which requires following pre-installed software: Flash Media Live Encoder (captures video, encodes it and streams it to the server), Adobe Media Server (receives the stream) and any network-stream-visualization software (VLC Media Player, Adobe Flash Player).

4.2.2.1 Camera placement

To achieve more accurate results, the camera must be carefully placed right above the gaming wheel on the minimal height of 0.5 meters above the wheel, so that the wheel itself can be fully covered withing camera perspective. Minimum placement errors are allowed:

- Position: $\pm$ 15 cm along x and z axis relative to wheel plane, $\pm$0.2 m above the wheel level.
- Rotation: $\pm$ 4° in Euler angles (pitch) along the wheel plane.

4.2.2.2 Minimal stream quality

Network stream minimal requirements:

- Bitrate: 600 kb/s.
- Latency: 70 ms.

Minimal requirements for the image, received from the network stream, can be described as:

- Resolution: 640x480 px.
- Codec: H264 – MPEG-4 AVC.
- Frame rate: 25 fps.
- Color space: ITU-R BT.601.

4.2.2.3 Roulette equipment requirements

The roulette wheel must fulfill such requirements, as:

- Size: the diameter of 55,88 cm and the height of 10,795 cm.
- Type: “European”, 37 pockets, including “zero” pocket and 36 default positions from 1 to 36.
- Sufficient wheel illumination, so that all the pockets are visible and can be easily recognized.

The bets table will not be needed, as it does not contain any valuable information for the algorithm.

4.2.3 Algorithm calculations flow

4.2.3.1 Original video frame

The entry point of the algorithm is receiving the current frame of the video transferred via network stream. The algorithm will keep the original image data through the whole workflow.

4.2.3.2 Masking

The roulette wheel itself contains segments that do not provide any valuable information for ball position calculations. For example, these elements can be bowl rim, wheel turret, the cone and others. The only valuable element of the wheel are pockets, since this is the only place where the ball is stopping.

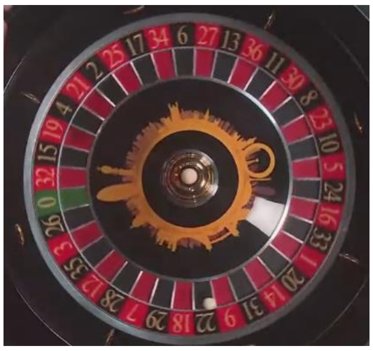


Figure 8. Example frame from the roulette video stream.

As seen on Figure 6, frames contain some unnecessary elements as listed before. These elements need to be culled out from the frame before further processing.

4.2.3.2.1 Intensity mask

Intensity masking supposes the multiplication of the original image with the predefined mask. In case of roulette, in order to cull everything, expect pockets, the mask can be a white donut-shaped figure with black background. As was described in section 3.1.1, the multiplication of the mask with the original image will result in the culled image.

A donut shape can be constructed using two ellipses. The first ellipse is larger than second one, therefore subtracting second ellipse from the first one will result in donut shape. The ellipse can be represented using the following equation:

$$ellipse(I(x,y),r) = 1 - \frac{dist\big(I(x,y)\big) \; - \; r}{ddxy(dist(x,y) - r)}$$

where $dist(I(x,y))$ is pixel distance from the corner of an image, r is ellipse radius and $ddxy(dist(x,y))$ is a combination of partial derivatives of the given pixel along x and y axis.

$$ddxy\big(I(x,y)\big) = |ddx(I(x,y))| + |ddy(I(x,y))|$$

The donut shape can be represented as an equation:

$$donut\big(I(x,y)\big) = ellipse(I(x,y),r_1) - ellipse(I(x,y),r_2)$$

where r_1 is bigger ellipse radius and r_2 is smaller ellipse radius. [2]

Having calculated the intensity mask, it is possible to perform frame culling. During this process, each pixel of the frame is iterated, and its value is multiplied with the value of the pixel on the same position in intensity mask. Every pixel on intensity mask has either a value of (255, 255, 255) or (0, 0, 0).

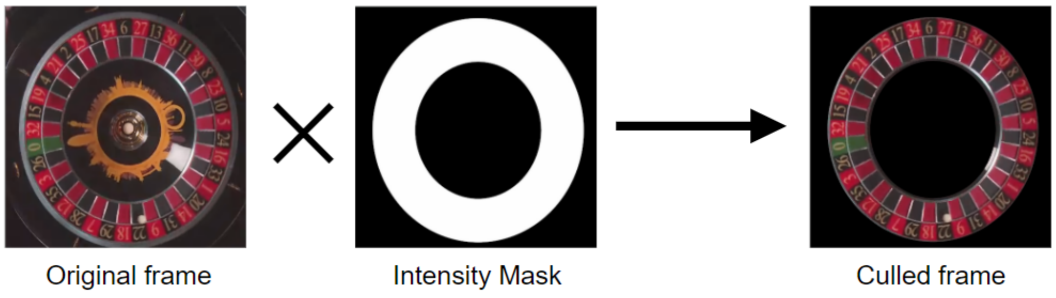


Figure 9. Visual representation of applying intensity mask to the frame.

4.2.3.3 Pockets positioning

The next step in the algorithm is defining pockets positions. Since the chosen roulette wheel model has 37 pockets – 37 separate sectors need to be determined. The roulette wheel is moving for the most part of the time, therefore positioning precision needs to be increased.

To achieve better precision, each pocket color must be sampled multiple times, however, the bigger the samples amount is, the slower is this algorithm step. For example, if the color is sampled 3 times instead of 1, the precision would increase 3 times, but the step would be $3^2 = 9$ times slower (will take 9 times longer to be performed).

Firstly, the "zero" pocket angle must be calculated, this will help to define wheel orientation.

Assuming that $\Delta r,\ \Delta g, \Delta b$ are channel differences between expected colors and sampled colors (i.e. the length of color difference vector), the total frame difference can be calculated using an equation:

$$d = \sum^{s} \sqrt{\Delta r^2 + \Delta g^2 + \Delta b^2}$$

where s is total samples count.

An accurate wheel orientation and pocket positions can be found calculating the total difference s times and choosing the smallest one. The "zero" pocket angle can be represented as:

$$\alpha_0 = \left(d_{best} + \frac{\pi}{n_p}\right) mod\ 2\pi$$

where d_{best} is best (smallest) frame difference and n_p is pockets count.

Each pocket can be separated from each other in the separation points, which can be calculated using the following equations:

$$\alpha_p = \alpha_0 + \frac{2\pi}{n_p * p}$$

where α_p is pocket angle and p is pocket index (from 1 to 36). The angle is calculated in radians.

$$x_p = \sin \alpha_p$$

$$y_p = -\cos \alpha_p$$

where x_p, y_p are separation point pixel coordinates.

Having calculated all 37 separation points, the wheel can now be separated into 37 equally sized sections.

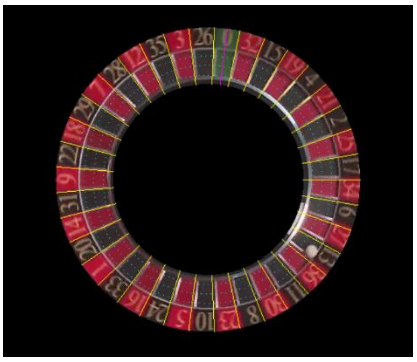


Figure 10. Separated wheel pockets.

4.2.3.4 Quantization

Before moving to the next steps, each pixel on the frame must be converted to discrete colors. This can be achieved using color quantization, which generally represents a process of reducing the distinct colors number in the frame. Quantization will help with segmentation and edges search in further steps.

All non-culled pixels withing the masked frame before performing color quantization, since it is only needed to quantize the pockets area. After that, the frame itself must be temporarily encoded into the byte array. This allows performing quantization using only bitwise and logical operations.

As soon as frame has been encoded into byte array, color quantization can be performed. Assuming that the final pixel can be of black (0), red (1), white (2) or green (3) color, pixel final color index can be calculated using the following algorithm (while iterating through all available pixels):

$$I_q[i * 3] = index(I_o[i] \ll 16\ OR\ I_o[i+1] \ll 8\ OR\ I_o[i+1])$$

where I_q is quantized frame, i is current pixel index in the iteration loop, $I[i]$ represents a pixel at given pixel index, I_o is original frame. "≪" is left-shift operator, that shifts left-hand operand left by the number of bits defined by its right-hand operand.

The result of this algorithm is an index (in range from 0 to 4) of a color. After receiving a new pixel color, it must be assigned to all non-culled pixels in the processed frame.

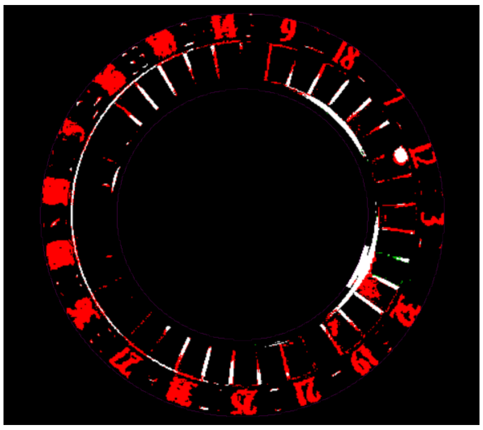


Figure 11. Color quantization performed on culled frame.

As seen on Figure 9, pixels on the quantized frame took one of 4 colors. Red color – for pocket numbers, white color – for edges, green color – for "zero", black color – for culled edges.

Image quality has decreased, however, pockets can still be visually distinguished. Result of the color quantization can be used during the edge detection. The main purpose of the color quantization is to eliminate the color diversity of the frame, as the ball has solid color, so its position determination will be easier.

4.2.3.5 Edge detection

Detecting the edges would allow to determine the approximate ball position. There are multiple ways to determine the edges on the quantized frame.

4.2.3.5.1 Threshold difference

One of the naïve ways of finding the edges – find the difference between similar thresholds. To find a threshold, a step operation can be performed either on one of the color channels of the frame or on the whole frame with the greyscale map applied on (described in sections 2.2.1 and 3.1.2).

The greyscale map can be applied to the frame using the following equation upon all non-culled frame pixels:

$$I(x,y) = dot((I(x,y).red, I(x,y).green, I(x,y).blue), (0.25, 0.5, 0.5))$$

where $(I(x,y).red, I(x,y).green, I(x,y).blue)$ and $(0.25, 0.5, 0.5)$ tuples can be considered as three-dimensional vectors, dot function can be represented as a dot product of two vectors:

$$dot(a,b) = \vec{a} * \vec{b} = \|\vec{a}\| * \|\vec{b}\| * \cos\theta$$

where θ is the angle between vectors.

The dot product allows to specify each channel proportion when applying greyscale map. As quantized frame has the red color as most common color, the red channel proportion needs to be lowered.

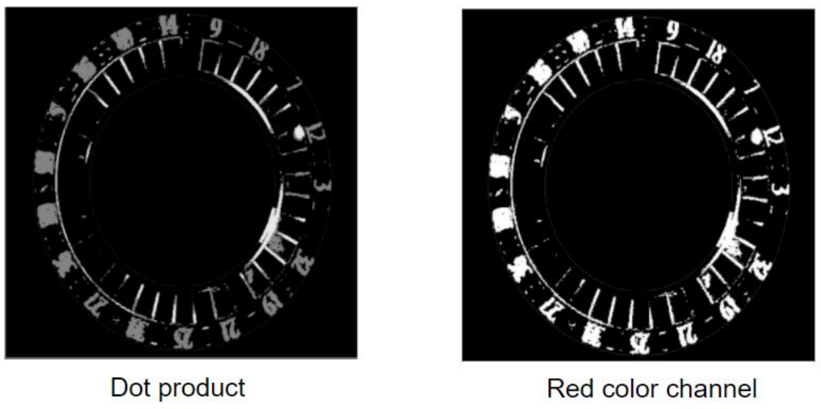


Figure 12. Threshold methods comparison.

Having acquired the intensity maps (basis frames), threshold difference can be calculated using the following equation:

$$I(x,y) = step(I(x,y), c_1) - step(I(x,y), c_2)$$

where c_1 and c_2 are threshold coefficients, i.e. the maximum pixel intensity for culling. [2] After multiple investigations, author had picked coefficients $c_1 = 0.17, c_2 = 0.8$ as best-fitting coefficients for calculating the threshold difference.

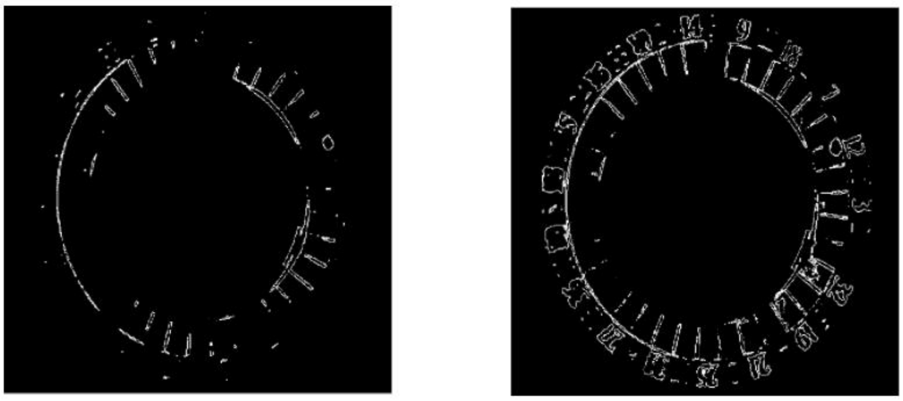


Figure 13. Threshold difference applied on different basis frames.

As can be seen on Figure 11, basis frame acquired using the dot product provides better predictions of the approximate ball position (a small circle on the right side of the frame), as there are less closed figures on the frame comparing to another frame, where all pocket numbers and separators were included.

4.2.3.5.2 Neighbor pixels method

Another possible solution for finding the edges is determining neighbor pixels for each non-culled pixel in the frame. The main idea of this method is to find non-culled neighbor pixels for each pixel and repeat this operation until the algorithm reaches the pixels which has no neighbors.

To find sufficient pixel neighbors, each pixel needs to be checked in four directions: along negative/positive x axis and along negative/positive y axis. If neighbor pixel has a value more than (0, 0, 0), i.e. not black color, it can be considered as sufficient neighbor pixel.

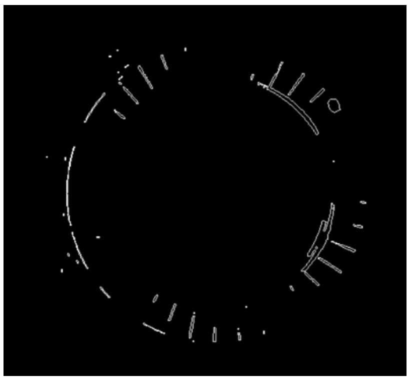


Figure 14. Neighbour pixels edge-seeking algorithm applied to the frame.

For further explanations, author will use threshold difference method using dot-product basis frame.

4.2.3.6 Hough transform

Having calculated the approximate ball position and acquired a frame with edges, it is possible to calculate more accurate ball position. Since the ball has circular shape, the main method that is going to be used is Hough transform, which is sufficient for finding primitive figures on the image such as lines, circles and ellipses in the frame.

A circle in the 2D space can be described by an equation:

$$(x-a)^2+(y-b)^2=r^2$$

where (a,b) vector is center of the circle, r is circle radius.

The ball has fixed radius, so it is not necessary to predict this parameter. For each point (x,y) on the ball circle, it is possible to define circle with a center in (x,y) and with a radius of r. A common intersection point of all such circles (Hough circles) will be the center of ball circle, i.e. the ball position.

To find all possible Hough circles along the estimated circle, the following equations will be helpful for finding the offset vector from the estimated circle center:

$$x_h=\frac{\Delta s}{i_h}*(-1)^n$$

$$y_h = \sqrt{r^2 - \left(\frac{\Delta s}{i_h}\right)^2} * (-1)^n$$

where x_h, y_h are offset vector coordinates relative to the center of estimated circle, r is average Hough circle radius, Δs is slope change, i_h is an offset vector, n is negativity parameter.

The slope change can be described using the following equation: [4]

$$\Delta s = \frac{r}{\sqrt{2}} + 1$$

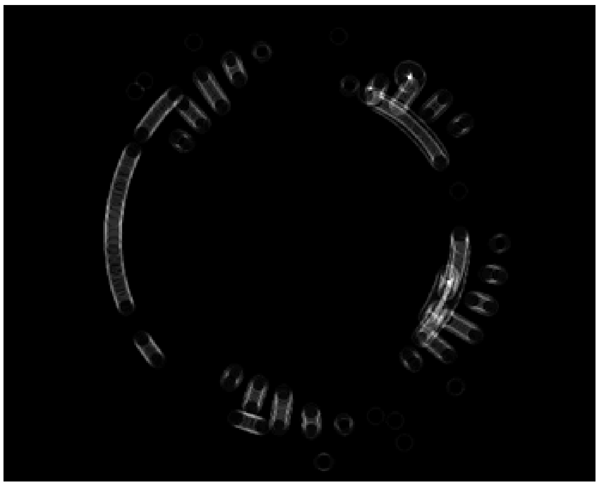


Figure 15. Hough transform applied on the edges frame. The ball position can be seen as a white spot in the top right corner.

Having defined the accurate ball position, the result can be now determined.

4.2.3.7 Result determination

The roulette wheel has fixed order of pockets deployed on it. For “European” roulette the pockets order can be defined as a set of numbers starting from “zero” pocket in clockwise direction:

$$\left\langle \begin{matrix} 0, 32,15,19,4,21,2,25,17,34,6,27,13,36,11,30,8,23,10,5,24,16,33,1,20,14,31, \\ 9,22,18,29,7,28,12,35,3,26 \end{matrix} \right\rangle$$

Pocket index (i_p) value lies in range from 1 to 37 (both inclusive).

The ball angle relative to the top part of the wheel can be calculated using the following equation:

$$\alpha_b = \left(\tan^{-1}\left(\frac{y_b - \frac{1}{2}}{x_b - \frac{1}{2}} \right) + 2\pi \right) mod\ 2\pi$$

where x_b, y_b are ball position coordinates.

The index of the pocket in the pocket numbers set can be calculated in the following way:

$$i_p = \left(\left(\frac{\gamma_p}{360} * n_p \right)\ mod\ n_p \right) + 1$$

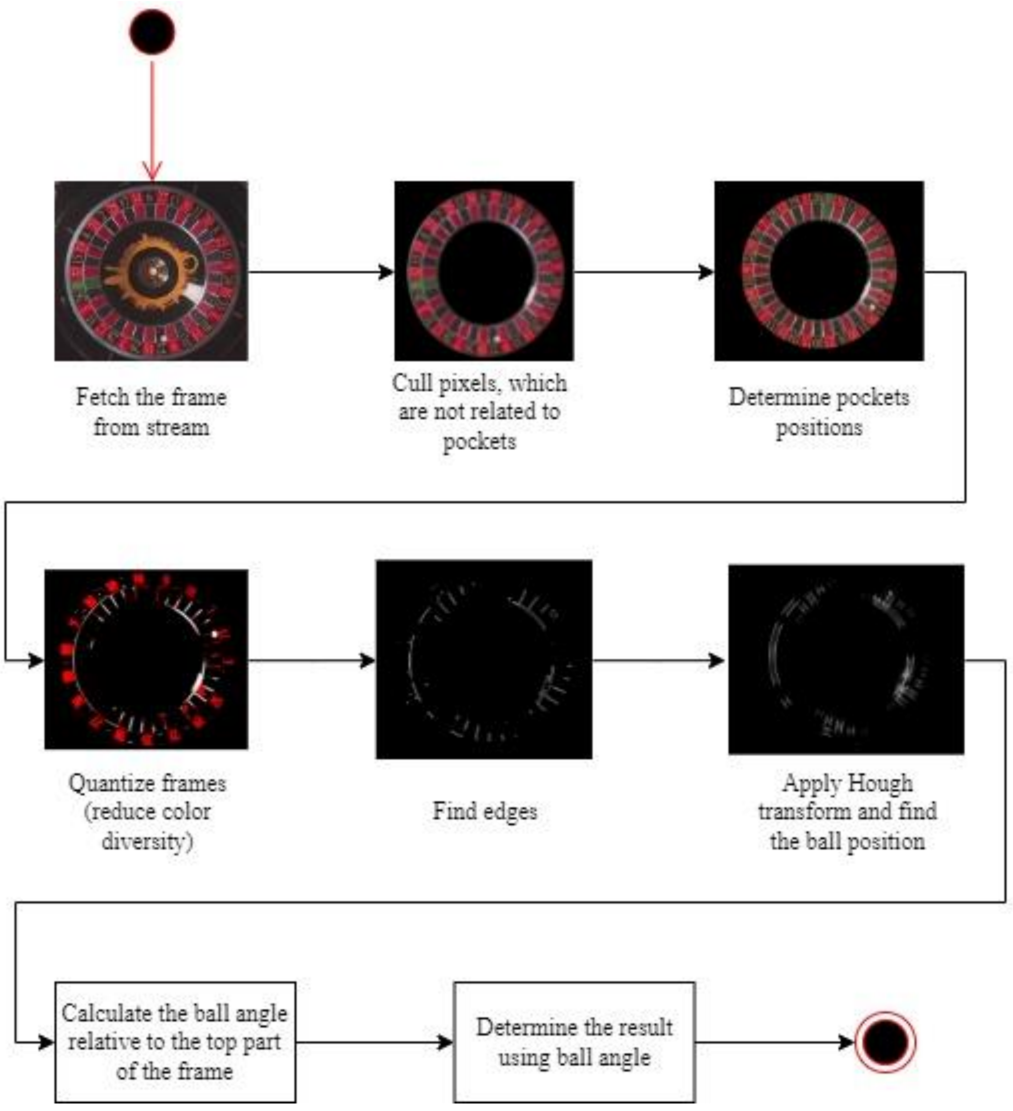
where γ_p is pocket-to-ball angle, n_p is pockets number.

Pocket-to-ball angle calculation method can be described as:

$$\gamma_p = \left(\frac{\alpha_b - \alpha_0}{2\pi * 360} + 360 \right) mod\ 360$$

where α_0 is “zero” pocket angle (see section 4.2.3.3).

4.2.4 Algorithm flow diagram



Fetch the frame from stream
Cull pixels, which are not related to pockets
Determine pockets positions
Quantize frames (reduce color diversity)
Find edges
Apply Hough transform and find the ball position
Calculate the ball angle relative to the top part of the frame
Determine the result using ball angle

Figure 16. Algorithm calculations flow represented as a flowchart.

4.2.5 Processing base

Each algorithm step can be performed either on GPU or CPU. Both can perform a variety of tasks, but the right processing base choice is crucial for overall algorithm performance.

Operations described in the most of steps can be performed directly on GPU and can be visualized using, for example, shaders, which are lightweight sets of instructions that describe how each pixel is rendered on the GPU side. These instructions are executed upon each pixel of the frame, which can seem unoptimized at the first time. However, one of the main concepts of the modern GPU is parallel processing, which makes them much more efficient in performing such sets of operations, than CPU. [2]

The “Masking” step, for example, can be directly performed on the GPU side, since the main purpose of this step is multiplying each pixel value with corresponding value from intensity mask.

On the other hand, the “Quantization” step, for example, should be performed on the CPU side, as it contains bitwise operations which are the part of basic logic, that CPU can process swiftly.

Every time GPU receives the frame on input, it has to sample each frame pixel in order to perform further operations on sampled pixels, therefore, as said before, “Quantization” step can be performed directly on CPU using bitwise operations, thus it will not require additional sampling and will save some part of GPU resources.

5 Tests

5.1 Testing platform

The internal testing application of Playtech company was chosen as a testing platform for the algorithm, as it has a base for results and effectiveness comparison (the main thesis goal is to find faster ways to recognize ball position), as well as it has integrated support of network streams.

The platform itself is written on WPF framework (a part of .NET Framework) using C# programming language and XAML (eXtensible Application Markup Language). The application has sufficient functionality for retrieving a statistical information for each algorithm step, as well as visualization functionality of each operation.

The algorithm was implemented using C# language and integrated into testing platform using internal methods.

5.2 Testing system specifications

Specifications of the platform that will be used for testing the algorithm can be described as:

- CPU: Intel® Core™ i5-7300 2.60 GHz
- RAM: 24 GB DDR4
- GPU: Intel® HD Graphics 620
- OS: Windows 10
- OS Build: 19042.1645

5.3 Input parameters for tests

Multiple streams were be taken for tests from the internal Playtech network. All streams meet minimal stream requirements which were described in section 4.2.2.2. Each stream has the spatial resolution of 640x480 px and average latency of 30 ms, as well as “European” wheel deployed in front of the camera.

The camera was placed 0.6 meters above the roulette wheel, with an offset of ~3 cm along x axis and ~1.2 cm along z axis relative to the wheel plane.

5.4 Testing method

The main testing strategy which author have chosen is quantitative research. Such strategy can be considered as numerical representation and manipulation of observations for the purpose of describing and explaining the phenomena that those observations reflect. [3]

All observations will be performed by an internal Playtech testing platform (described in section 5.1).

Each algorithm step takes t_i time, where i is step index (from 1 to 7, as the total amount of steps is 7). As t_i value is varying over time, step performance is represented as an average of all measurements done during single ball spin. Therefore, the intermediate flow performance during the current frame can be described as:

$$t = \sum t_i$$

It is important to note, that t is how much time does algorithm take to calculate the ball position in the single (current) frame. Cumulative algorithm performance during single game round is defined as the average of all records of intermediate flow performance.

Each step is a part of algorithm pipeline, i.e. the step itself cannot be removed from the pipeline, as its output is used in the next steps.

5.5 Test structure

Tests are split into 10 separate cases. Each case contains different streams of different quality (streams quality fulfil minimum requirements), therefore each case represents different production setups.

During each testing case, 30 roulette game rounds are played. During each round, dealer throws the ball with different strength and in different directions (clockwise or anti-clockwise).

Each case has also different environment and roulette gaming equipment. For example, the roulette wheel can have different pocket color (green, red), the room where wheel is deployed can have different lighting setups.

Each round takes about 30 seconds, and during this period of time, algorithm will perform calculations and save intermediate results, which will be analyzed in further steps.

6 Results and analysis

6.1 Algorithm performance

After performing 10 separate test cases and playing a total of 300 roulette rounds, tests results can be represented as a table of time measurements, which shows overall algorithm performance:

Table 1. Test cases intermediate results.

	Average result determination time	**The slowest step**	**Correct result determination ratio**	**Failure ratio**
Case #1	25.2728 s	4.8109 ms	97.14 %	2.1 %
Case #2	24.7774 s	5.27 ms	97.59 %	2.08 %
Case #3	24.9866 s	5.2762 ms	98 %	1.98 %
Case #4	24.3081 s	4.4326 ms	96.98 %	1.9 %
Case #5	25.5072 s	5.1930 ms	97.33 %	2.11 %
Case #6	24.6733 s	4.9134 ms	97.4 %	2.14 %
Case #7	25.7436 s	5.5492 ms	98.01 %	1.9 %
Case #8	25.0675 s	4.5674 ms	97.99 %	1.95 %
Case #9	24.7217 s	5.2023 ms	97.19 %	2.04 %
Case #10	25.3212 s	4.8749 ms	97.48 %	1.92 %

Note: each value in the table row represents an average of the measurements taken during playing 30 roulette rounds.

Table columns of the Table 1 can be described as:

- **Average result determination time**: the average time for which the algorithm calculates and determines the final round result, i.e. **how many seconds have passed since the start of the round** when the algorithm has determined the final result.
  Note: the ball approximate position can be calculated starting from about 10-15 seconds from the start of the round.

- **The slowest step**: the longest period of time in which one of the steps performed calculations within one frame.
- **Correct result determination ratio**: the percentage of correctly determined results out of the total amount of rounds.
- **Failure ratio**: the percentage of failed determinations out of the total amount of rounds.

6.1.1 Results analysis

Before analyzing tests results, it is important to note, that comparison base (previous internal algorithm) is a property of Playtech company, so that its source is closed and only performance data is provided in research purposes within this thesis.

As seen on the Table 1, it takes in average about 24-25 seconds for algorithm to determine the final result for 1 roulette round. Comparing with the previous algorithm (that software is currently using), which takes about 35-40 seconds to determine the result (the result is determined after the end of the round), the performance grow can be estimated in average as 35%, as well as the result is now determined before the end of the round, which leads to the reduction of the delays in production.

Tests results have shown that the slowest step of the algorithm is pocket positioning step (described in section 4.2.3.3), which took in average the maximum of 5.27 milliseconds to calculate the intermediate result. This can be caused by the multiple color sample processes performed during this step, as the samples amount value has been set to 12, since this value turned out to be the most optimal.

The correct results determination ratio has shown to be in average about 98%. This parameter shows that in 98% of total the result was correctly defined, but in other 2% the calculated result and real result were different. Previous algorithm has the correct determination ratio value of about 74%. The result accuracy growth can be estimated as 25%.

The failure ratio average value is about 2%, meaning that in 2% of all rounds the algorithm failed to determine results because of lack/relatively low quality of analysis data. Such data includes: pockets visibility, ball color, ball size, ball color and pockets color difference, etc. Comparing to the previous algorithm, which had the failure ratio of 13%, the failure regression can be described as 650%.

6.1.2 Failure cases analysis

The algorithm failed to determine the result in average 2% of all cases, which is not critical for production, since all dealers can type in the result manually in case of errors/flaws. The most common reason of a failure is environment lighting setup. For example, metallic glare of the roulette wheel part can cause flaws in thresholding and quantization steps.

Another parameter, which contains specific number of failures, is correct result determination ratio. This parameter has shown, that in about 2% of all cases the algorithm determined wrong result. It can be caused by various reasons. For example, the camera positioning error is crucial in determining the pockets position, and even the smallest deviation can lead to the wrong result determination.

6.1.3 Separate step performance analysis

Each step has difference execution time, since each step performs different operations. Steps as “Masking” and “Edge detection” consist of basic mathematical/logical operations, therefore they show the best performance among other steps. On the other side, “Quantization” step has shown the worst performance, as this step consists of multiple sampling operations made over all frame pixels.

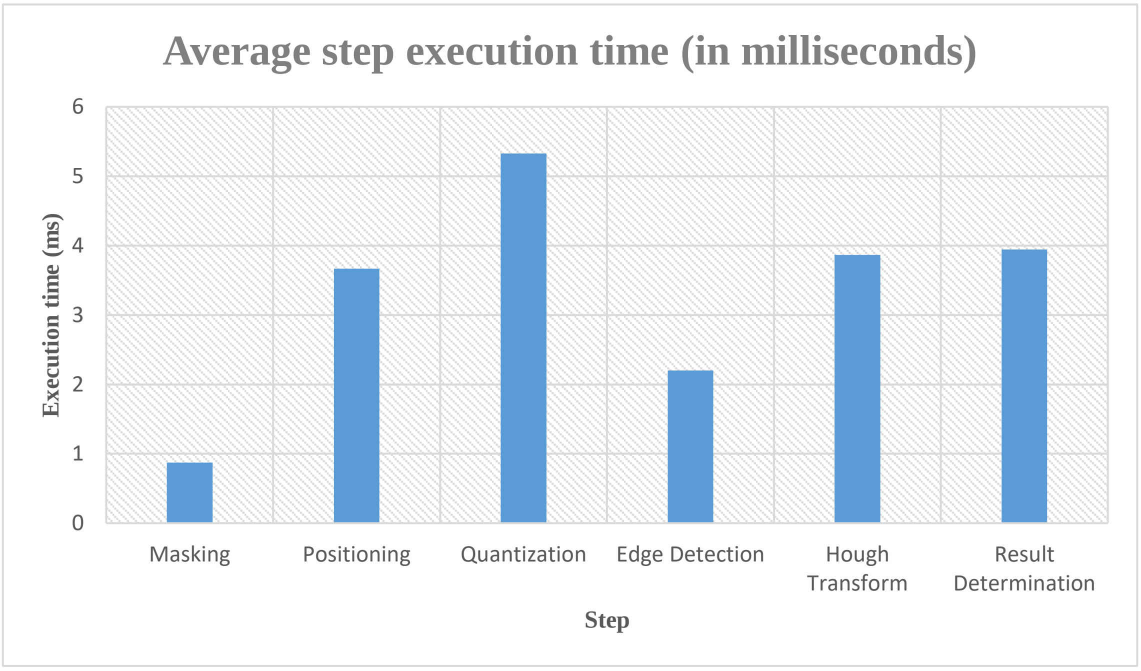


Figure 17. Average step execution time among 300 roulette rounds.

As can be seen on Figure 17, as described before, the slowest step is “Quantization”. The most efficient steps are “Masking” and “Edge Detection”. “Hough Transform”, “Positioning“ and “Result Determination” steps have relatively medium performance indicators.

7 Summary

Having commited multiple tests and analysed the result, it can be stated that the main goal was achieved. The prototype works 35% faster than currently used Playtech software internal algorithm, the result determination failure chance has dropped in 6.5 times and the total accuracy has raised by 25% in average.

On the other hand, the algorithm still can determine the wrong result or fail to define the ball position. The pockets positioning step can be optimized, as it is the slowest step so far.

There is much space for further algorithm development, including steps optimization, new ways of quantizing stream frames (as for now the quantization process includes bitwise operations, which require converting frames to the byte sequences), preventing failures (reducing the total chance of failure by, for example, exploring and implementing new ways of thresholding). Possible improvements also include implementing self-training model (for example, neural network) which can possibly help avoiding failure cases (by, for example, detecting metallic glares on roulette wheel).

References

[1] C.Solomon, Toby Breckon, "Fundamentals of Digital Image Processing". [Web-material]. Available: http://share.its.ac.id/pluginfile.php/827/mod_resource/content/1/referensi/ebooksclub.org_Fundamentals_of_Digital_Image_Processing_A_Practical_Approach_with_Examples_in_M.pdf. [Accessed on 28 03 2022]

[2] "The book of shaders", [Web-material]. Available: https://thebookofshaders.com. [Accessed on 02 04 2022]

[3] S. Sukamolson, "Fundamentals of quantitative research", [Web-material]. Available: https://www.researchgate.net/profile/Vihan-Moodi/post/What_are_the_characteristics_of_quantitative_research/attachment/5f3091d0ed60840001c62a27/AS%3A922776944787456%401597018576221/download/SuphatSukamolson.pdf. [Accessed on 15 04 2022]

[4] J. Illingworth, J. Kittler, "The Adaptive Hough Transform", [Web-material]. Available: https://ieeexplore.ieee.org/document/4767964. [Accessed on 12 04 2022]

[5] M. Bailey, S. Cunningham, "Graphics Shaders: Theory and Practice", Boca Raton: CRC Press, 2011.

[6] "CYCLOPS Roulette Displays", [Web-material]. Available: http://www.setproduction.fi. [Accessed on 03 05 2022]

[7] "European Roulette", [Web-material]. Available: https://www.casinogamespro.com/roulette/european-roulette. [Accessed on 11 04 2022]

[8] "Stream live media (RTMP) with Adobe Media Server", [Web-material]. Available: https://helpx.adobe.com/adobe-media-server/dev/stream-live-media-rtmp.html. [Accessed on 11 04 2022]

[9] "Supreme Roulette Series", [Web-material]. Available: https://www.gambee.eu/supreme-roulette/. [Accessed on 03 05 2022]

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

[1] The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.